REMARKS

Claims 1-9, 11, 13-23, 28, 29, 31-33, 35, 36, 38-40 and 42-57 were pending in the application. Claims 11 and 19 have been amended. The amendments are supported in the specification by Figures 1A, 1B, and 1C. Claims 58-59 are newly submitted. The additional claims are supported in Appendix A (page 1) and in the specification on page 4, lines 13-18; page 7, lines 7-9; and page 8, line 12. Reconsideration of the Action mailed April 8, 2004 is respectfully requested in view of the amendments to the claims and the following remarks.

Claims 1, 2, 5-7, 9, 28, 29, 35, 36, 42, 45, and 48 were rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Pat. No. 5,469,538 to Razdow in view of U.S. Pat. No. 5,404,428 to Wu. In rejecting the claims, the Examiner, citing column 9, lines 8-26 of Razdow, states that Razdow teaches a method in a computer program for maintaining dependencies among a set of objects each having a value, the set of objects including an object A and an object B, the method of maintaining dependencies comprising: when the value of object A is a function of the value of object B and the value of object B changes, marking object A as dirty and not recomputing the value of object A until object A is queried for a value, when the value of object B changes, invalidating the dependents of object B and all of their further dependents. The cited passage of Razdow actually reads as follows:

> "When the user modifies (or deletes) an expression that has been previously entered, the editor 10 modifies the expression and places the modified expression in the correct location in the document 12. The expression compiler 14 generates a new value of the modified expression. Next, the expression compiler 14 automatically examines the numerical dependency graph and marks as "out of date" all expressions that depend on the modified expression. Next, each node that has been marked out of date in turn recalculates itself (with the assistance of the numerical interpreter 18). In performing each recalculation, the node looks up the numerical dependency graph 16 to find the new values of any recalculated nodes (i.e., expressions). When a node is recalculated, the numerical interpreter 18 automatically updates the document 12. Accordingly, the document 12 is always the most current representation of the nodes of the numerical dependency graph 16.

> Thus, all expressions in the document 12 are in fact consistent with all antecedent expressions upon which they depend."

Nothing in the cited passage indicates that the value of object A is not recomputed until object A is queried for a value, as recited in claim 1. In fact, the cited passage describes a system in which "each node that has been marked out of date in turn recalculates itself" (column 9, lines 15-17) as the next step after being marked out of date, and the node that is marked out of date hence does not wait to be queried for value before recomputing itself. For at least this reason, claim 1 and its dependent claims, are allowable.

Further as to claim 1, the Examiner combines Razdow with Wu and states that Wu teaches severing dependencies among the dependents of object B and all of their further dependents and causing each invalidated observer-only object to recompute its value by querying the values of the objects from which the observer-only object depends, citing column 9, lines 1-47 in Wu. The cited passage actually reads as follows:

> "Each derived item declined in the system in implemented embodiments contains references, in the form of direct function calls for evaluating other items which have validity flags. In this way, when a view model attribute is modified by the application program, derived items dependent upon that attribute change, are invalidated during the change. No calculations are performed at the time of an attribute change. Once the device pipeline requests one of the invalid derived items, then the calculations may be performed. This is performed by descending the acyclic graph until valid item(s) are reached, and then calculating invalid item(s) back up the acyclic graph, clearing invalid flags for the item(s) until the item requested is reached, whose invalid flag is then cleared. The acyclic directed graph describes the derived items that become invalid upon a change in value to any view model attribute. It also describes the dependencies of each derived item on other items, and hence an optimal evaluation traversal for any derived item requested by a pipeline."

While the cited passage describes the process for invalidating dependents, a careful review shows nothing that indicates that the dependencies, defined in Wu by the acyclic graph that is mentioned in the cited passage, are ever severed or even changed. As a result, a system

described in Wu is one in which the values of objects may change (and become invalid) while the relationships between the objects are fixed. For at least this reason, claim 1 and its dependent claims, are allowable.

In rejecting claim 7, the Examiner finds the limitation "identifying the objects upon which a given object depends as those objects into which the given object passed itself as a requestor during execution of a compute method of the given object" in Razdow, column 8, lines 21-43 and column 11 lines 39-61. The cited passages actually read as follows:

> "If a new expression is entered by the user, the expression compiler 14 creates a new node in a numerical dependency graph 16. For example, if the expression entered was "n:=2" then the expression compiler 14 calculates, in conjunction with a numerical interpreter 18 (sometimes also called a numerical computational engine), the value of the expression (in this case, the simple calculation that n has the value 2), and creates a node in the numerical dependency graph 16, where the node has the value 2, and the arc coming from the node represents the variable "n". As another example, if the expression entered was "y:=2+3", then a node is created that represents this expression. The value of the expression is calculated as 5. Thus, the node has the value 5, and the arc coming from the node represents the variable "y". As a further example, if the expression entered was "y:=2+a" then a node is created that represents this expression. The value of the expression is calculated only if " a" has a value in an expression in the document 12 that is above or to the left of the expression "y:=2+a". If "a" is so defined, then the node has an arc coming into the node represents the variable "a". (The structure of the numerical dependency graph 16 is discussed in more detail below with reference to FIG. 3a.)" [column 8, lines 21-43]

> "In contrast to the numerical dependency graph 16, where the nodes represent programs that calculate numerical values for entered expressions and where the arcs are variable names that represent the corresponding numerical variables, the nodes of the SDG 26 represent expressions qua expressions and the arcs represent variable names that represent the corresponding expressions. For example, if the user enters the following expressions:

x:=5; (11)

y:=x-1; (12)

and then entered the expression

(x+1).y= (13)

(the "=" sign in the representative embodiment of the present invention signifying that this is to be evaluated as a numerical expression) then the numerical dependency graph 16 of FIG. 3a would be produced. The result, the number "24" would be displayed at expression (13) as follows:

(x+1).y=24 (13a)" [column 11 lines 39-61]

A careful review of the cited passages shows nothing that teaches an object that passes itself as a requestor during the execution of a compute method. For at least this reason, claim 7 and its dependent claims are allowable.

Claim 11, 13-18, 31, 32, 38, 39, 49 and 50-53 were rejected under 35 U.S.C. 103(a) as being unpatentable over U.S. Pat. No. 6,272,672 B1 to Conway in view of Wu.

Claim 11, as amended, recites a method for changing objects in which "the changed object [has] objects depending directly on the changed object and objects depending indirectly on the changed object through an object different from the changed object."

Conway describes a system that "employs a quasi-static constraint-network model" (column 9, lines 43-44), which "consists of component instances interconnected by wires" (column 14, lines 21-23). These "wires" are static and are never severed as a result of any changes to the values of the components. In Conway's system, alterable dependencies exist between components and flow objects, and these dependencies can be severed as a result of a change in the value of a given flow object (figures 7-11, column 25, lines 22-67, column 26, lines 1-35). However, all of the dependent components of a given flow object in Conway are directly dependent on that flow object, and no component can depend on a flow object indirectly, that is through another component or through another flow object. Because no dependencies can

exist among a set of flow objects, Conway does not teach a method for changing objects, in which objects can depend indirectly on the changed object as recited in claim 11. For at least this reason, claim 11 and its dependent claims are allowable.

Claims 19-23, 33, 40, 54-57 were rejected under 35 U.S.C. 102(e) as being anticipated by Conway.

Claim 19, as amended, recites limitations corresponding to those of claim 11 and is therefore allowable for at least the reasons set forth above in reference to claim 11.

Please charge the amount of $36.00 for excess claim fees to our deposit account, number 06-1050. Please apply any other charges or credits to deposit account 06-1050.

Respectfully submitted,

Date: 30 Jun 04

Hans R. Troesch
Reg. No. 36,950

Fish & Richardson P.C.
500 Arguello Street, Suite 500
Redwood City, California 94063
Telephone: (650) 839-5070
Facsimile: (650) 839-5071

50218559.doc